ARL-TR-7353 • JULY 2015

US Army Research Laboratory

An Experimental Exploration of the Impact of Sensor-Level Packet Loss on Network Intrusion Detection

by Sidney C Smith and Robert J Hammell II

Approved for public release; distribution is unlimited.

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-7353 ● JULY 2015

US Army Research Laboratory

An Experimental Exploration of the Impact of Sensor-Level Packet Loss on Network Intrusion Detection

by Sidney C Smith
Computational and Information Sciences Directorate, ARL

Robert J Hammell II
Department of Computer and Information Sciences, Towson University

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
July 2015	Final	August 2012-March 2015

4. TITLE AND SUBTITLE
An Experimental Exploration of the Impact of Sensor-Level Packet Loss on Network Intrusion Detection

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Sidney C Smith and Robert J Hammell II

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-CIN-S
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-7353

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
primary author's email: <sidney.c.smith24.civ@mail.mil>.

14. ABSTRACT
In this report we consider the problem of sensor-level packet loss (SLPL) as it applies to network intrusion detection. We explore 2 research questions: 1) Is there sufficient regularity in SLPL to allow an algorithm to be developed to model it? and 2) Is the impact of SLPL on network intrusion detection performance sufficiently regular to allow a formula to be developed that will accurately predict the effect? We developed and validated the Pcapreplay program, which allowed us to characterize the manifestation of SLPL. We conducted experiments using Pcapreplay and Snort to explore the impact of SLPL. We graphed and analyzed this impact against our previous theoretical work. We conducted experiments using Pcapreplay and Snort to measure the impact on network intrusion detection. We graphed the alert loss rate against the packet loss rate. We compared these graphs to our previous theoretical work. We used nonlinear regression analysis to produce a formula with r-squared and reduced chi-squared values close enough to 1 for us to answer both of our research questions in the affirmative.

15. SUBJECT TERMS
packet loss, network intrusion detection, software, Snort, Tcpdump, Pcapreplay

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Sidney C Smith
Unclassified	Unclassified	Unclassified	UU	36	19b. TELEPHONE NUMBER (Include area code) 410-278-6235

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

INTENTIONALLY LEFT BLANK.

1. Introduction

As we consider the impact of sensor-level packet loss (SLPL) on network intrusion detection (NID), we observe that NID depends upon the sensor being able to see the traffic between the adversary and the target. General packet loss is very common on the Internet. The Transmission Control Protocol is specifically designed to account for general packet loss and uses it as a barometer to gauge the available bandwidth of a connection.[1] In this report we are not interested in general packet loss because those packets cannot cause a compromise since they will never reach the target. We are interested in what we call detection-relevant packet loss (DRPL). DRPL occurs when packets reach the target but fail to reach the sensor software for analysis. Since the sensor cannot detect what it cannot see, DRPL must have a negative impact on the sensor's ability to detect malicious activity. Based upon the large amount of work that has been done to reduce or eliminate DRPL on NID, we infer that the negative impact of DRPL on NID is well known. This report is part of a larger effort to understand, predict, and model the impact of DRPL on NID. In our previous theoretical work,[2] we divided DRPL into network, host, and sensor levels. In this report, we will focus on DRPL at the sensor level. SLPL is defined as any packets that are processed by the host operating system but are not processed by the analysis software.

The focus of this report is to answer 2 research questions concerning the manifestation of SLPL and the impact of SLPL on NID: 1) Is there sufficient regularity in SLPL to allow an algorithm to be developed to model it? and 2) Is the impact of SLPL on NID performance sufficiently regular to allow a formula to be developed that will accurately predict the effect? We discovered that the manifestation of SLPL is not random but is very similar to the capped algorithm of the packet dropper we developed in our previous theoretical work.[2] We also discovered that the impact of SLPL is not random but is very similiar to both the results of the capped-by-packet algorithm of the packet dropper and a sigmoid function. When we used regression techniques to compare these algorithms, each of them had R^2 and χ^2_v values very close to 1. These discoveries allow us to answer both of our research questions in the affirmative.

In Section 2 we provide an overview of the existing research, which is heavily focused upon eliminating packet loss at the sensor level. In Section 3 we discuss the Pcapreplay program, which we developed to explore SLPL, the dataset that we

used in these experiments, the experiment to characterize SLPL, and the experiment to measure the impact of SLPL. Section 4 describes the results of our experiments and our regression analysis of these results, and Section 5 summarizes our results and discusses future work.

2. Background

In previous research[2] we considered the theoretical impact of packet loss on NID. We divided the potential for packet loss among the network, host, and sensor level. We also defined network-level packet loss as any packet that reaches the target but fails to reach the network segment where the sensor is located. Additionally, we defined host-level packet loss as any packet that reaches the network interface of the sensor but is not presented to the analysis software. SLPL is defined as any packet that is presented to the analysis software but is not processed. The various level of packet loss may be seen in Fig. 1. SLPL is represented by bit bucket C.

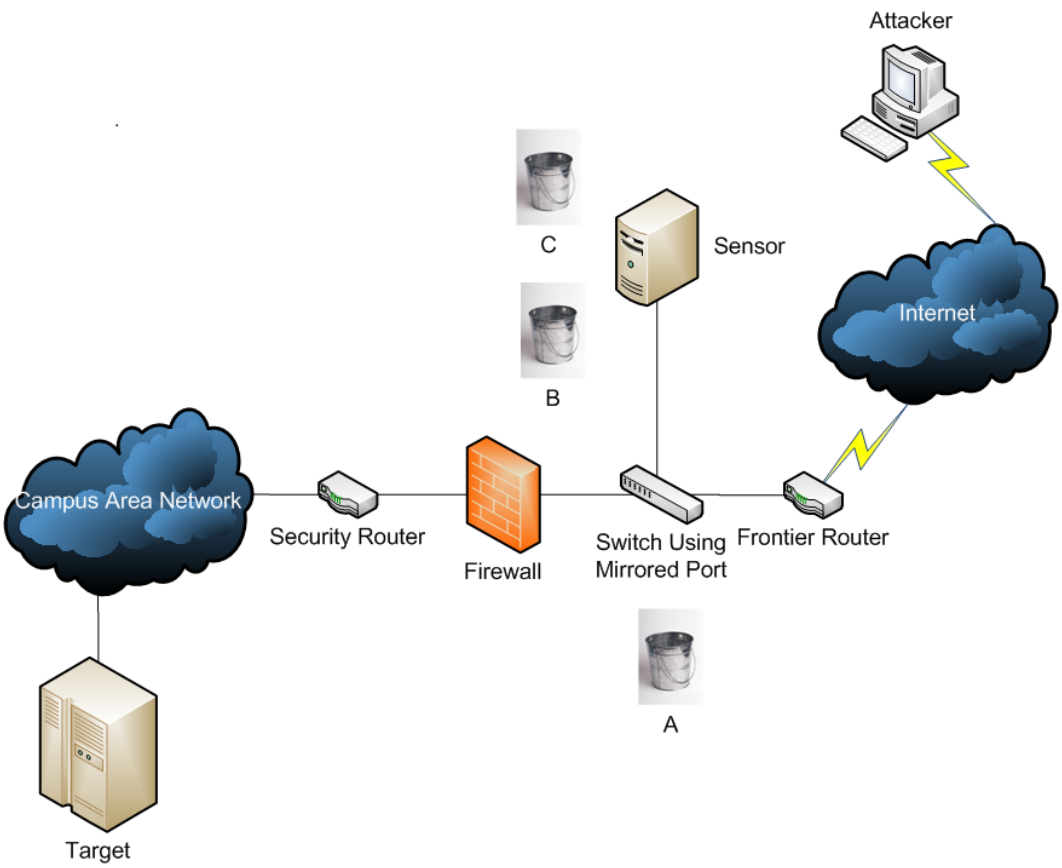

Fig. 1 Detection-relevant packet loss diagram

We constructed the packet dropper, which abridges datasets according to several different algorithms we implemented to emulate our theories about how packets may be lost. We also visualized the characterization of this packet loss by graphing the network traffic from 2.5 min of traffic from the 2009 Cyber Defense Exercise(CDX)[3] dataset, which we describe in detail in Section 3.2, as it was abridged at 25% packet loss by each of the packet dropping algorithms.[2]

Although work has been done to reduce or eliminate packet loss, there has been little work done to understand and model packet loss and its impact on NID. We were able to glean several insights into packet loss from the work done to reduce or eliminate packet loss. In Schaelicke and Freeland's work on characterizing the sources and remedies for packet loss in NID, they obtained some improvements in packet loss rates by increasing the level of optimization employed in the compiler.[4] They also found that significant improvements in packet loss could also be achieved by pruning the sensor rule set.[4] Additionally, Salah, in his work on improving snort performance under Linux, discovered that setting the netdev_budget, which is a kernel configuration parameter in the New Application Programming Interface, to a low number greatly decreases packet loss.[5] He concluded that this occurred primarily because pulling packets from the network and analyzing the packets are bound by available central processing unit (CPU) cycles and not buffer memory. Allocating the bulk of the CPU time to the sensor application enables the system to process more packets. Additionally, the packets that are dropped are dropped very early in the process, preventing CPU cycles from being wasted processing packets that will only be dropped later.[5] Futhermore, in their work on hash-based load balancing, Kim et al. successfully employed multiprocessing techniques to increase the throughput of the sensor and reduce packet loss.[6] In their work on achieving flow-level controllability in NID, Song et al. point to the software crash as yet another cause of packet loss.[7] Finally, in their work on adapting the intrusion detection system model to load characteristics under Internet Protocol version 6, Wei et al. proposed breaking the detection task into 3 units: the Data Acquisition Unit, the Adapt Load Characteristic Analysis Unit, and the Collaborative Analysis and Control Center.[8]

Revisiting bit bucket C in Fig. 1, we expected that the resource consumption of the sensor application itself will contribute the greatest component of packet loss at this level. We theorized that we could simulate this effect through a cyclic function,

such as a sine wave that would model a process gathering resources, performing its task, and releasing the resources.[2]

3. Approach

Our approach to answering our research questions requires several building blocks. First, we will need some method to simulate and isolate SLPL; we will use the Pcapreplay program developed for this effort, as described in Section 3.1. We will also need NID software for our experiment; we will use Snort[9] for this purpose. Last, we will need network capture data in Tcpdump[10] format; we will use portions of the CDX[11,12] 2009 dataset, described in greater detail in Section 3.2, for this purpose. We will compare this data to data generated using the packet dropper[2] we developed in a previous effort. In addition to the tools descibed above, our approach to answer our first research question is detailed in Section 3.3, and our approach to answering our second research question is detailed in Section 3.4.

3.1 Pcapreplay

Answering our research questions requires that we isolate and measure SLPL. If we configured Snort to read from the network stack, it would be impossible to know whether the packets were dropped because the network stack could not process them or because Snort could not process them in time. To isolate the sensor layer, we will pipe packets in Tcpdump format to Snort on standard input. Snort will report the number of packets that it dropped; however, that will provide no insight into which packets were dropped. Snort will either capture packets or it will analyze packets, but it will not do both. To gain insight into how these packets are lost, we will need to have some way to record which packets are lost. To do this, we have written the Pcapreplay tool, which reads a file in Tcpdump format using Libpcap[13] and writes the packets in Tcpdump format at any multiple of the original speed. Packets that cannot be written in time are written to a bit bucket file. We can then analyze the bit bucket file to gain insight into how the packets are lost in order to determine if there is sufficient regularity in SLPL to allow an algorithm to be developed to model it. We understand that in many ways this is similar to the Tcpreplay[14] command, which will take packets from a file saved in Tcpdump format and replay them on the network at any multiple of the original captured speed.

Algorithm 1 provides the psuedo code for the core of the Pcapreplay application. The trickiest part of this algorithm is the fudge factor, which was included because

the library functions sleep[15] and usleep[16] are guaranteed to sleep for the specified time; however, they may oversleep. We added the fudge factor to account for the system oversleeping.

Algorithm 1 Core Pcapreplay algorithm

Require: $fileStartTime = 0$
  while get next packet **do**
    $fileTime \leftarrow packetTime$
    $currentTime \leftarrow gettimeofday()$
    if $fileStartTime = 0$ **then**
      $fileStartTime \leftarrow fileTime$
      $startTime \leftarrow currentTime$
      $elapseStartTime \leftarrow currentTime$
      $initialDifference \leftarrow currentTime - fileTime$
    end if
    $elapseTime \leftarrow currentTime - startTime$
    $effectiveTime \leftarrow fileStartTime + elapseTime * accelerator$
    if $filetime + fudge \geq effectiveTime$ **then**
      $packetsWritten \leftarrow packetsWritten + 1$
      $sleepTime = (fileTime - effectiveTime)/accelerator$
      $sleep(sleepTime)$
      $timeNow = gettimeofday()$
      $newEffectiveTime \leftarrow timeNow - initialDifference$
      if $newEffectiveTime > effectiveTime + sleeptTime$ **then**
        $fudge = newEffectiveTime - effectiveTime - sleepTime$
      end if
      $writepacket$
      $flushstream$
    else
      $packetsDropped \leftarrow packetsDropped + 1$
      write packet to bit bucket
    end if
  end while

To validate that Pcapreplay is actually replaying packets at the specified multiple of the orginal speed, we will use the script described in Algorithm 2 to conducted several trials measuring the time it takes for Pcapreplay to replay 1 h of network traffic. A full description of the capabilities and options of the Pcapreplay tool is in the Appendix. We will use the script described in Algorithm 3 to conduct several trials measuring the time it takes for Tcpreplay to replay the same 1 h of network traffic. If Pcapreplay is working correctly, we should find that the results are very similar. These results may be found in Table 1. Computing the value of R^2 for

Pcapreplay, using Formula 1, we get 0.9999, which is the same R^2 value that we get for Tcpreplay and close enough to 1 for us to conclude that Pcapreplay is actually replaying packets at the correct speed.

$$R^2 = \frac{\text{SS}_{\text{reg}}}{\text{SS}_{\text{tot}}}. \tag{1}$$

$$\text{SS}_{\text{reg}} = \sum_{i=1}^{n} (\hat{y}_i - \bar{y}_i)^2. \tag{2}$$

$$\text{SS}_{\text{tot}} = \sum_{i=1}^{n} (y_i - \bar{y})2. \tag{3}$$

Algorithm 2 Validate Pcapreplay shell script

```
#!/bin/sh
speed=1
while [ ${speed} -lt 2048 ]; do
        time\
        pcapreplay -x ${speed} -b /dev/null -w /dev/null $*
        speed=`expr ${speed} \* 2`
done
```

Algorithm 3 Validate Tcpreplay shell script

```
#!/bin/sh
speed=1
while [ ${speed} -lt 2048 ]; do
        time tcpreplay -x ${speed} -i eth1 $*
        speed=`expr ${speed} \* 2`
done
```

3.2 Dataset

Annually, the National Security Agency/Central Security Service conducts an exercise pitting teams from the military academies of the United States and Canada against teams of professional network specialists to see who can best defend their network.[3] In their paper, "Toward Instrumenting Network Warfare Competitions to Generate Labeled Datasets", Sangster et al.[11] describe their efforts to collect and label traffic from the 2009 competition. We were able to obtain this data from https://www.itoc.usma.edu/research/dataset/.

Table 1 Pcapreplay and Tcpreplay validation values

Speed	Expected	Pcapreplay	Tcpreplay
1	3600.000	3564.012	3564.962
2	1800.000	1781.646	1784.305
4	900.000	890.749	892.991
8	450.000	444.901	447.348
16	225.000	222.259	224.431
32	112.500	111.214	112.976
64	56.250	55.191	57.315
128	28.125	27.688	29.349
256	14.063	13.424	15.619
512	7.031	6.787	8.755
1024	3.516	3.457	5.340
2048	1.758	1.742	3.891

This dataset was chosen because one of the files captured by gator010, 20090421.14.ftrm, contains traffic that is consistent enough for us to be able to see the influence of packet loss on the network traffic. We will be using the same 2.5 min of network traffic that was used in our previous research.[2] This will allow us to compare our experimental results with our previous theoretical results.

3.3 Characterizing SLPL

To answer our first research question (Is there sufficient regularity in SLPL to allow an algorithm to be developed to model it?), we will establish the null hypothesis that sensor-level packet loss is sufficiently irregular, as to be effectively considered random. We have plotted the effect of random packet loss on this dataset in Fig. 2. To characterize the sensor-level impact, we will use Pcapreplay to replay the CDX dataset, piping it to Snort version 2.9.3.1 using the community rules set from August 2013. Some trial and error will be necessary to discover the Pcapreplay speed setting that will produce approximately 25% packet loss. We constructed a shell script that would conduct trials starting at 2x, continuing until 32768x in a geometric progression, i.e., 2^n, using a shell script like the one shown in Algorithm 4. This script should allow us to quickly discover the correct multiplier to achieve 25% packet loss. Once we discover the appropriate speed factor, we can graph the network traffic in the bucket.pcap file. If the SLPL is random, we should see the abridged traffic line beneath and almost identical to the orginal traffic line. If we see this kind of relationship, we must accept the null hypothesis. If there is some regularity to SLPL, the abridged traffic line will show some difference to the original traffic line beyond

a simple reduction. This may be seen in the graph of the traffic from the sinusoidal dropping algorithm in Fig. 2. If we see this kind of relationship, we must reject the null hypothesis.

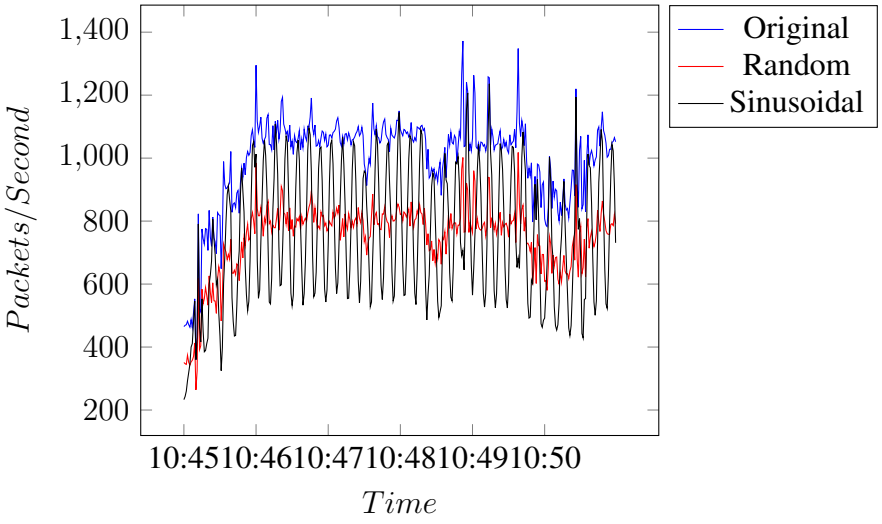

Fig. 2 Manifestation of 25% packet loss generated by the packet dropper and measured in packets/second

Algorithm 4 Shell script to automate repeated runs of Pcapreplay

```
#!/bin/sh
X=2
while [ $X -lt 65536 ]; do
        mkdir log${X}x
        pcapreplay\
                -x ${X}\
                -w -\
                -L log${X}x/replay.log\
                -b log${X}x/bucket.pcap $* |
        snort\
                -c ~/rules/etc/snort.conf\
                -r -\
                -l log${X}x 2> log${X}x/snort.out
        X=`expr $X \* 2`
done
```

3.4 Measuring the Impact of SLPL

To answer our second research question (Is the impact of SLPL on NID performance sufficiently regular to allow a formula to be developed that will accurately

predict the effect?), we will establish the null hypothesis that the impact of sensor level packet loss is sufficiently irregular as to be effectively considered random. In an attempt to prove the null hypothesis, we will run the CDX gator010 data through Pcapreplay at various speeds, piping the output to Snort. We will be able to use the shell script from Algorithm 4 with slight modifications to allow for different progressions based upon the results we see from the initial trial in an attempt to obtain a good spread through the packet loss rate (PLR) domain of 0% to 100%. The replay.log file will contain the PLR, and the snort.out file will contain the number of alerts. A control run of Snort against the dataset without Pcapreplay will tell us the total number of alerts, which we may use to calculate the alert loss rate (ALR). Plotting the PLR verses the ALR will allow us to compare the results against our results from the theoretical exploration.[2] We should be able to use curve fitting and regression analysis techniques to discover a relationship and measure the fit. If we are able to discover a relationship and obtain an R^2 value close to 1, we will reject the null hypothesis and may safely conclude that the impact of SLPL on NID performance is sufficiently regular.

4. Results

We will divide our results according to our research questions. In Section 4.1, we will describe the results of our efforts to observe how packet loss manifests itself at the sensor level. In Section 4.2, we will describe the results of our efforts to observe the impact this packet loss has on NID.

4.1 Sensor-Level Packet Loss Manifestation

When we ran our first trial, as described in Section 3, we discovered a speed setting on our hardware that generated 25% packet loss. In Fig. 3, the blue line represents the original traffic, and the red line represents the traffic abridged by Pcapreplay at 25% packet loss. The traffic pattern does not resemble the random or function-influenced traffic patterns from our previous research,[2] seen in Fig. 2. Instead, it resembles the patterns produced by the capped algorithms of the packet dropper (see Fig. 4 for comparison). In Fig. 4, the blue line represents the original traffic and the red line represents the traffic as abridged by the packet dropper using the capped algorithm.

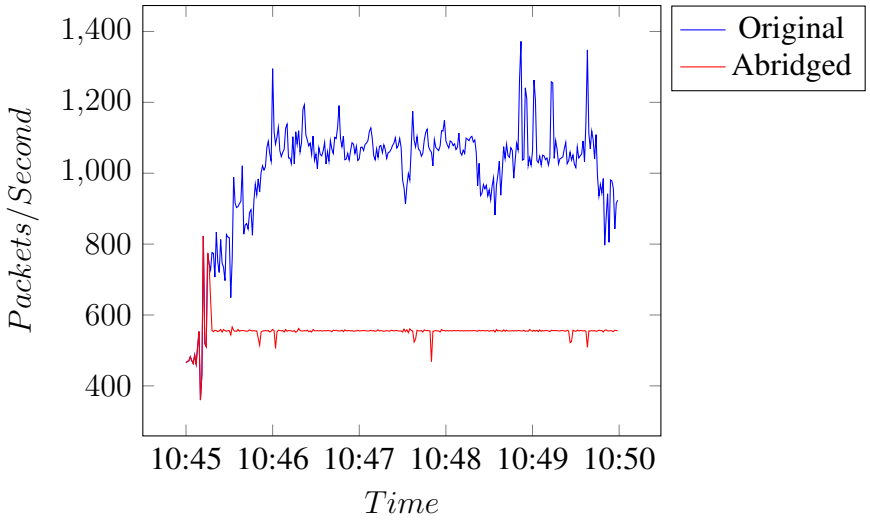

Fig. 3 Manifestation of 25% packet loss generated by Pcapreplay and measured in packets/second

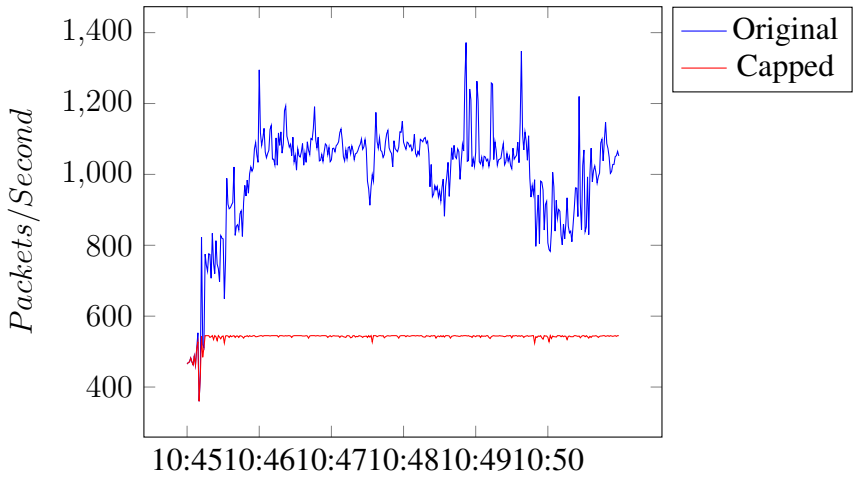

Fig. 4 Manifestation of 25% packet loss generated by the capped algorithm and measured in packets/second

There is a caveat in that the resources of the system used to conduct the experiments have an impact on the shape of the traffic graph. Fig. 5 demonstrates that as the size of the system increases, the traffic graph flattens. The red line represents the results of the experiment run on desktop-class equipment. This red line shows massive oscillation between an upper and lower bound centered around the cap. The yellow line represents the results of the same experiment run on a small server. Some variance is visible in the yellow line; however, the line is much flatter. The black line,

which shows very little variance, represents the results of the same experiment run on a large server.

We observed another difference between the graphs produced by Pcapreplay and the graphs produced by the packet dropper. The line representing packets per second produced by the packet dropper immediately plateaus at the cap. The line representing packets per second produced by Pcapreplay continues to follow the normal traffic flow until it drops and quickly settles into the cap. These observations have little bearing on the current research; however, they may have significant influence in future work as we attempt to validate the results of the packet dropper.

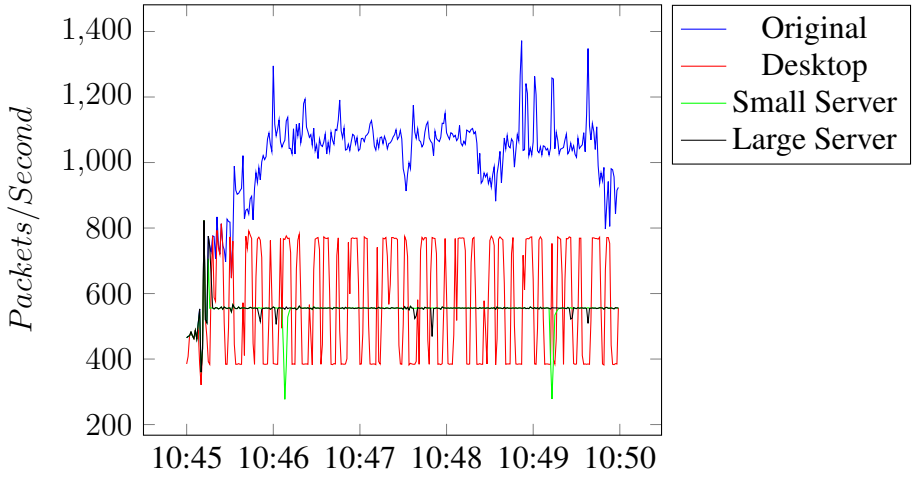

Fig. 5 Comparison of the manifestations of 25% packet loss generated by Pcapreplay on different hardware and measured in packets/second

4.2 Impact on the Performance of NID

We created several versions of the shell script described in Section 3 and executed them to collect 4 trials. Each trial consisted of several runs of Pcapreplay, replaying the CDX dataset described in Section 3.2, piped to snort version 2.9.3.1 using the August 2013 rules set. We were able to collect the number of packets lost from the Pcapreplay log and the number of alerts detected from the output of Snort. A control run of Snort against the dataset without using Pcapreplay revealed that there were 5,218,144 packets in the dataset and 275 alerts. We used this information to compute the PLR and ALR for each run of the various trials. Given the similarity in the manifestation of the packet loss to the capped algorithms of the packet dropper, we expect to find a graph similar to Figs. 6 or 7. Figure 6 is the graph of the PLR

versus the ALR when using the capped-by-packets algorithm of the packet dropper. Figure 7 is the graph of the PLR versus the ALR when using the capped-by-bits algorithm of the packet dropper. In Fig. 8 we see the experimental results.

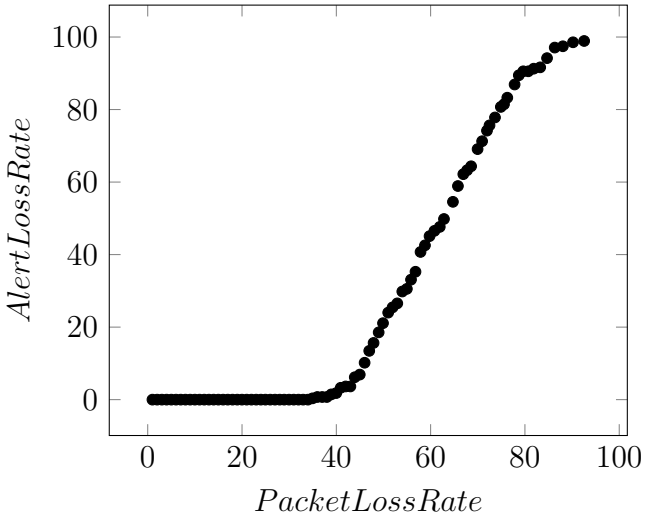

Fig. 6 PLR vs. ALR for CDX gator010 abridged with the capped-by-packets algorithm

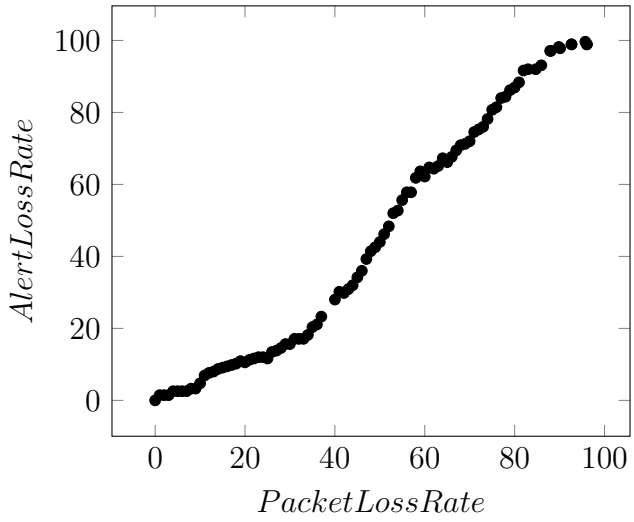

Fig. 7 PLR vs. ALR for CDX gator010 abridged with the capped-by-bits algorithm

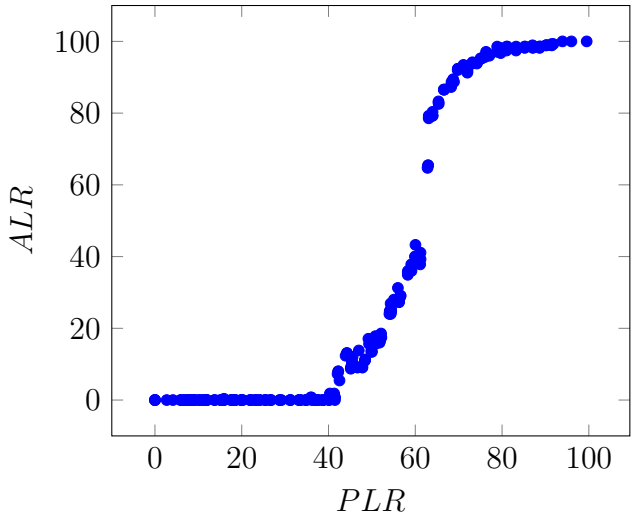

Fig. 8 PLR vs. ALR for CDX gator010 abridged with Pcapreplay

Notice that the graph generated by the packet dropper using the capped-by-packets algorithm appears to be very similiar to our results obtained using Pcapreplay. This confirms Salah's finding that this process is CPU and not memory bound.[5] In Fig. 9 we have plotted the results on the same graph to show this similarity. If we were to treat the ALR of the packet dropper as the predictor $\hat{y}$ and the ALR of Pcapreplay as y, we could compute R^2 to determine how well the packet dropper predicts the results of Pcapreplay. This assumes that x or the PLR is the independent variable, which is the model that we are seeking. The problem is that the true independent variable in the Pcapreplay experiments is the speed multiplier, and PLR is a function of that. This means that we cannot use the complete datasets from these experiments. We must use a subset of the datasets where the PLR values are the same. In Table 2 we see the selected data points, and in Table 3 we see several key values. Using Formula 1 to compute R^2, we get a value of 0.96, which would be close enough to 1 to declare that the packet dropper and Pcapreplay are a good fit if this were a linear regression. Our observation of the graph reveals that this is most likely a nonlinear relationship resembling a sigmoid function (see Fig. 10). Spiess and Neumeyer report that R^2 is inadequate for judging the goodness of fit for nonlinear models.[17] They found that the Akaike Information Criterion (AIC), bias-corrected AIC (AIC_c), Bayesian Information Criterion (BIC), residual variance (resVar), chi-square (χ^2), and reduced chi-squared (χ^2_v) were better indicators of goodness of fit for nonlinear regression. These are computed using Formulas 5,

6, 7, 8, 9, and 10, respectively,[17] where $ln(L)$ is the maximum log-likelihood, p is the number of parameters in the model, and σ^2 is the variation. Since the packet dropper uses an algorithm rather than a formula, we set $p = 3$ because the algorithm depends upon 3 indepedent variables. It turns out that the AIC, AIC$_c$, and BIC are very good for comparing the fit to different forumlas; however, they do not indicate very much standing alone. The χ^2_v value may be used to judge the goodness of fit for nonlinear regression, where the closer χ^2_v is to 1, the better the fit. In this case χ^2_v is 1.1785, indicating that we have a good fit.

$$\ln(L) = 0.5(-N(\ln 2\pi + 1 - \ln N + \ln \sum_{i=1}^{n} x_i^2)). \tag{4}$$

$$\text{AIC} = 2p - 2\ln(L). \tag{5}$$

$$AIC_c = \text{AIC} + \frac{2p(p+1)}{n-p-1}. \tag{6}$$

$$\text{BIC} = p\ln(n) - 2\ln(L). \tag{7}$$

$$\text{resVar} = \frac{\text{RSS}}{n-p}. \tag{8}$$

$$\chi^2 = \sum_{i=1}^{n} \frac{(y_i - f(x_i))^2}{\sigma_i^2}. \tag{9}$$

$$\chi^2_v = \frac{\chi^2}{v}. \tag{10}$$

Table 2 Packet dropper and Pcapreplay comparison values

PLR	PD ALR	PR ALR	e	e^2	$(y - \bar{y})^2$	$(\hat{y} - \bar{y})^2$
5	0.00	0.00	0.00	0.00	1561.95	1561.95
10	0.00	0.00	0.00	0.00	1561.95	1561.95
15	0.00	0.00	0.00	0.00	1561.95	1561.95
20	0.00	0.00	0.00	0.00	1561.95	1561.95
25	0.00	0.00	0.00	0.00	1561.95	1561.95
30	0.00	0.00	0.00	0.00	1561.95	1561.95
35	0.36	0.00	-0.36	0.13	1561.95	1533.34
40	1.82	0.00	-1.82	3.31	1561.95	1421.54
45	6.91	8.73	1.82	3.31	948.29	1063.57
50	21.09	13.45	-7.64	58.31	679.49	339.69
55	30.55	28.00	-2.55	6.48	132.75	80.57
60	45.09	40.00	-5.09	25.92	0.23	31.02
65	54.55	79.27	24.73	611.44	1580.16	225.72
70	69.09	92.36	23.27	541.62	2792.29	874.35
75	80.73	95.27	14.55	211.57	3108.20	1697.91
80	90.55	96.73	6.18	38.21	3272.50	2987.44
85	94.18	98.18	4.00	16.00	3441.03	2987.75
90	98.55	98.91	0.36	0.12	3526.88	3483.82
96	99.64	100.00	0.36	0.13	3657.64	3613.79

Table 3 Key values to describe the relationship between the packet dropper and Pcapreplay

Name	Value
$\bar{\mathbf{x}}$	50
$\bar{\mathbf{y}}$	39.52
SSE	1516.56
SS_{tot}	35635.06
SS_{reg}	29328.22
SS_{res}	1516.56
R^2	0.96
$\ln(L)$	-14452.8
AIC	28911.54
AIC_c	28913.14
BIC	28914.38
resVar	94.78512
χ^2	19
χ^2_v	1.1875

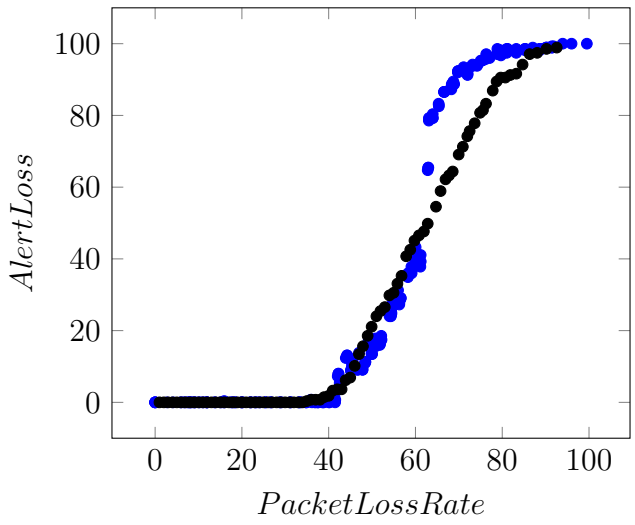

Fig. 9 PLR vs. ALR for CDX gator010 abridged with the packet dropper using the capped-by-packets algorithm and abridged by pcapreplay

We will now look more closely at the Pcapreplay data to determine if there exists a relationship $y = f(x)$, where y is ALR and x is PLR. Observing Fig. 8, we see that the relationship between PLR and ALR is nonlinear. The growth in ALR is slow at the start, and at the end of the PLR domain is an S-shaped pattern. This is the behavior of a sigmoid function illustrated in Formula 11 and Fig. 10. We will use the general logistic function in Formula 12 as the model for our nonlinear regression,

where A is the lower asymptote, K is the upper asymptote, B is the growth rate, v affects near which asymptote maximum growth occurs, Q depends on the value of y when $x = 0$, and M is the time of maximum growth. We were able to use the solver in Microsoft Excel to discover values for A, K, B, v, Q, and M. These values are shown in Table 4 along with the values for several other variables, including the R^2 and χ^2_v.

$$y = \frac{1}{(1 + e^{-x})}. \tag{11}$$

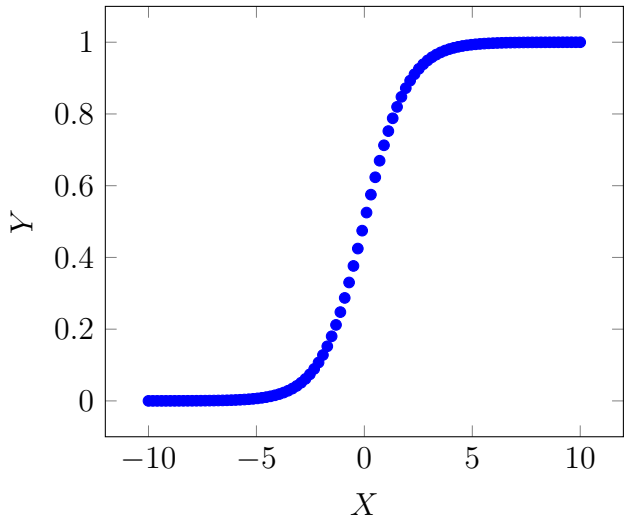

Fig. 10 Sigmoid nonlinear regression model

$$y = A + \frac{K - A}{(1 + e^{-B(x-M)})^{\frac{1}{v}}}. \tag{12}$$

Table 4 Key values of the regression analysis of the impact of SLPL

Name	Value
A	0
K	100
B	200.948
v	2.88963
Q	0.36356
M	68.4576
SSE	2765.56
$\bar{y}$	35.78
$\bar{x}$	48.56429
SSR	385175.22
SS_{yy}	386765.96
SS_{xx}	146326.0840484140
R^2	0.9958896459
$\ln(L)$	-1829.221623
AIC	3670.443247
AIC_c	3670.803762
BIC	3691.32708
resVar	11.81862199
χ^2	240
χ^2_v	1.025641026

5. Conclusions

Through this research we were able to answer each of our research questions in the affirmative. We determined that SLPL is sufficiently regular to allow an algorithm to be developed to model it. We discovered that the capped-by-packets algorithm of the packet dropper, developed in a previous work,[2] models SLPL in this dataset with R^2 and χ^2_v values very close to 1. We were able to determine that the impact of SLPL on NID performance is sufficiently regular to allow a formula to be developed that will accurately predict the effect, and we defined a formula that predicts the effect on this dataset with R^2 and χ^2_v values very close to 1. We claim that the existance of the packet dropper algorithm that models SLPL accurately for this dataset and the formula that predicts the impact of SLPL on NID performance for this dataset

demonstrate the affirmative answers to our research questions. We make no claim that this algorithm or formula is the correct algorithm or formula to model SLPL or the impact of SLPL in general.

The ultimate goal is to discover a general function $y = f(x)$, where y is ALR and x is PLR. This function will allow us to accurately predict the impact of packet loss on NID. To achieve this goal, a study of the combined effect of packet loss across all 3 layers should be conducted upon the foundation that has been laid by studying each layer individually. The packet dropper algorithm needs to be refined and validated based upon the findings of the combined study. Replaying a dataset at several multiples of its original speed is a time–consuming process even when it may be completely automated. The CDX dataset that we used was originally 105 h long. Replaying the dataset in an exponential progression (i.e., 2^n) would require 210 h. Often this technique does not produce enough data points, and the process would take even longer. Some of the datasets studied in our previous research are 7 weeks long and would be completely impractical to study by replaying them at several multiples of the original speed. Having a validated packet dropper, which can process the dataset in minutes rather than days, would greatly benefit the research. With a validated packet dropper, it may be possible to analyze several datasets and generate and validate a general function.

6. References

1. Stevens WR. Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. Freemont (CA): Internet Engineering Task Force; 1997 Jan. RFC No.: 2001.

2. Smith S, Hammell R, Parker T, Marvel L. A theoretical exploration of the impact of packet loss on network intrusion detection. In: Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2014 15th IEEE/ACIS International Conference; 2014 Jun; Las Vegas, NV. p. 1–6.

3. Mazmanian A. CDX pits NSA hackers against service academies. Vienna (VA): FCW; 2014 Apr 10 [accessed 2015 Mar]. http://fcw.com/articles/2014/04/10/cyber-defense-exercise.aspx.

4. Schaelicke L, Freeland JC. Characterizing sources and remedies for packet loss in network intrusion detection systems. In: Workload Characterization Symposium, 2005. Proceedings of the IEEE International; 2005; Austin, TX. p. 188–196.

5. Salah K, Kahtani A. Improving snort performance under linux. IET communications. 2009;3(12):1883–1895.

6. Kim NU, Park MW, Park SH, Jung SM, Eom JH, Chung TM. A study on effective hash-based load balancing scheme for parallel nids. In: Advanced Communication Technology (ICACT), 2011 13th International Conference; 2011; Gangwon-Do, South Korea. p. 886–890.

7. Song B, Yang W, Chen M, Zhao X, Fan J. Achieving flow-level controllability in network intrusion detection system. In: Software Engineering Artificial Intelligence Networking and Parallel/Distributed Computing (SNPD), 2010 11th ACIS International Conference; 2010; London, UK. p. 55–60.

8. Wei C, Fang Z, Li W, Liu X, Yang H. The ids model adapt to load characteristic under ipv6/4 environment. In: Wireless Communications, Networking and Mobile Computing, 2008. WiCOM'08. 4th International Conference; 2008; Dalian, China. p. 1–4.

9. Roesch M. Snort: Lightweight intrusion detection for networks. In: LISA; Vol. 99; 1999; Seattle, WA. p. 229–238.

10. Jacobson V, Leres C, McCanne S. The tcpdump manual page. Berkley (CA): Lawrence Berkeley Laboratory; 1989.

11. Sangster B, O'Connor T, Cook T, Fanelli R, Dean E, Adams WJ, Morrell C, Conti G. Toward instrumenting network warfare competitions to generate labeled datasets. In: Proc. of the 2nd Workshop on Cyber Security Experimentation and Test (CSET09); 2009; Montreal, Canada.

12. U.S. military academy wins NSA's 14th annual cyber defense exercise. Ft. Meade (MD): National Security Agency Central Security Service; [date unknown; accessed 2015 Mar]. https://www.nsa.gov/public_info/press_-room/2014/U_S_Military_Academy_ Wins_NSAs_14th_Annual.shtml.

13. Jacobson V, Leres C, McCanne S. Libpcap. Berkley (CA): Lawrence Berkley Laboratory; 1994.

14. Turner A, Bing M. Tcpreplay: Pcap editing and replay tools for *nix. Boston (MA): 2005 [accessed 2015 Mar]. http://tcpreplay.synfin.net.

15. Sleep - sleep for the specified number of seconds. In: Chapter 3 of the Linux Programmerś Manual; Raleigh (NC): Red Hat, Inc.; 2014.

16. Usleep - sleep some number of microseconds. In: Chapter 3 of the Linux Programmerś Manual; Raleigh (NC): Red Hat, Inc.; 2014.

17. Spiess AN, Neumeyer N. An evaluation of r2 as an inadequate measure for nonlinear models in pharmacological and biochemical research: a Monte Carlo approach. BMC Pharmacology. 2010;10(1):6.

INTENTIONALLY LEFT BLANK.

Appendix. Pcapreplay Linux Online Manual Page

NAME

pcapreplay − Replays a libpcap file with a given acceleration.

SYNOPSIS

pcapreplay [options] [pcapfile] ...

-L	--logfile	set the file where dianostic info is written.
-b	--bitbucket	set the bit bucket file.
-h	--help	print usage information.
-r	--readfile	set the readfile.
-v	--version	print the version informaiton.
-w	--writefile	set the writefile.
-x	--accelerate	set acceleration factor.

DESCRIPTION

Pcapreplay concatenates the files in the pcap format given on the command line and writes them at the speed in which they were captured with a user provided acceleration factor. Packets that cannot be written in time may optionally be placed in a bit bucket file.

EXAMPLE(S)

The following command will replay the capture.pcap file at 5 times the orginal speed to standard output over a pipe to snort. Packets that cannot be written in time will be written to the bucket.pcap file.

$ *pcapreplay −x 5 −b bucket.pcap -w - capture.pcap | snort*

SEE ALSO

libpcap(3), tcpdump(8)

AUTHOR(S)

Sidney C. Smith, Computational Information Sciences Directorate, US Army Research Laboratory

List of Symbols, Abbreviations, and Acronyms

ACRONYMS:

ALR	alert loss rate
CDX	Cyber Defense Exercise
CPU	Central Processing Unit
DRPL	detection–relevant packet loss
NID	network intrusion detection
PLR	packet loss rate
SLPL	sensor–level packet loss

MATHEMATICAL SYMBOLS:

R^2	the coefficient of determination
SSE	sum of the squares of error SSE $= \sum_{i=1}^{n}(y_i - \hat{(y)}_i)^2$
SS_{tot}	total sum of squares $\text{SS}_{\text{yy}} = \sum_{i=1}^{n}(y_i - \bar{y})^2$
SS_{reg}	sum of squares regression $\text{SS}_{\text{reg}} = \sum_{i=1}^{n}(\hat{y}_i - \bar{y}_i)^2$
SS_{res}	residual sum of squares $\text{SS}_{\text{res}} = \sum_{i=1}^{n}(\hat{y}_i - y_i)^2$
$\ln(L)$	maximum log-likelihood $\ln(L) = 0.5(-N(\ln 2\pi + 1 - \ln N + \ln \sum_{i=1}^{n} x_i^2))$
AIC	Akaike Information Criterion AIC $= 2p - 2\ln(L)$
AIC_{c}	bias-corrected AIC $\text{AIC}_{\text{c}} = \text{AIC} + \frac{2p(p+1)}{n-p-1}$
BIC	Bayesian Information Criterions BIC $= p\ln(n) - 2\ln(L)$
resVar	residual variance resVar $= \frac{\text{RSS}}{n-p}$
χ^2	chi-square $\chi^2 = \sum_{i=1}^{n} \frac{(y_i - f(x_i))^2}{\sigma_i^2}$
χ_v^2	reduced chi-squared $\chi_v^2 = \frac{\chi^2}{v}$
σ_i^2	the variance $\sigma_i^2 = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2)}{n-1}$

MATHEMATICAL OPERATORS:

$\bar{y}$ the bar over the variable indicated the mean value for that variable

$\hat{y}$ the hat over the variable indicates the value predicted by the regression function

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
2 (PDF)	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL IMAL HRA MAIL & RECORDS MGMT
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
1 (PDF)	DIR USARL RDRL CIN S S SMITH

INTENTIONALLY LEFT BLANK.